Nombre del proyecto
Automatización de facturas vehiculares

Profesor	Mgt. Gladys María Villegas Rugel
Materia	Proyecto Integrador
Alumnos	Grupo 1: Andrea Fernanda Morán Vargas Pedro José Vidal Orús
Fecha	26 septiembre 2025

Resumen del Proyecto
Fase de Preparación y Procesamiento de Datos 1. Análisis Exploratorio de Datos (EDA) 2. Pipeline de Limpieza de Datos 3. Feature Engineering Avanzado 4. Estrategias de Balanceamiento 5. Data Augmentation 6. Partición Estratificada de Datos 7. Pipeline de Preprocessing

Actividad	Fecha
Automatización de facturas vehiculares	26 septiembre 2025

Contenido

1. Análisis Exploratorio de Datos (EDA)

1.1 Exploración Inicial Completa

Dado que actualmente no contamos con un dataset propiamente dicho, pues, en nuestro proyecto los datos se obtienen al extraer la información desde facturas emitidas de múltiples fuentes (etapa que está en construcción), hemos utilizado un Dataset, perteneciente a dos grupos con información definido: "COMPLETA" e "INCOMPLETA", el primero corresponde a facturas reales de Venta de Vehiculos y el segundo a facturas de repuestos con datos en su mayoría incompletos.

- **Primeras y últimas observaciones**

```
  FECHA_DOCUMENTO                                      DIRECCION  \
0      29/12/2023  Dir. Matriz: AVENIDA ATAHUALPA SN y RIO GUAYLL...
1      21/10/2009  Av. de los Granados E11-67 y Las Hiedras, Quit...
2       28/6/2021                          Av. Indoamerica Km 3 ½

  MODELO_HOMOLOGADO_ANT  SUBSIDIO     AÑO  SUBTOTAL   CLASE    TOTAL  \
0                   NaN       NaN     NaN       NaN     NaN      NaN
1                   NaN       NaN  2009.0  25810.96  CAMION  30400.0
2                   NaN       NaN  2022.0  20535.71  CAMION  23000.0

  CILINDRAJE                                      MODELO  ...       RUC  \
0        NaN                                         NaN  ...       NaN
1  3900 C.C.                          HD72 CHASIS CABINADO  ...  1,79E+12
2       2771  HOWO ZZ1047D3414C145 AC 2.8 2P 4X2 TM DIESEL  ...  1,89E+12

  COMBUSTIBLE  EJES    TIPO      IVA  \
0         NaN   NaN     NaN      NaN
1      DIESEL   NaN  CAMION  1331.90
2      DIESEL   NaN  CAMION  2464.29

                                   CONCESIONARIA TONELAJE  \
0                                                    NaN      NaN
1      HYUNDAI DEL ECUADOR HYUNDEC SOCIEDAD ANONIMA     2.80
2  VEHICENTRO VEHICULOS Y CAMIONES CENTRO SIERRA ...      3.6

          VIN_CHASIS PAIS_ORIGEN  ETIQUETA
0                NaN         NaN  COMPLETA
1  KMFGA17BP9C900561     ECUADOR  COMPLETA
2  LZZ5BADC7NF000057       CHINA  COMPLETA

[3 rows x 30 columns]
Últimas 3 filas del dataset:
   FECHA_DOCUMENTO                                        DIRECCION  \
24       19/5/2022       Av. Carlos Julio Arosemena Km 3.5, Guayaquil
25       13/5/2025  AV. RIO COCA E8-73 y PARIS / AV. SAN LUIS 670 ...
26       18/9/2024  WHYMPER N28-39 Y AV. FRANCISCO DE ORELLANA, Quito

   MODELO_HOMOLOGADO_ANT  SUBSIDIO     AÑO  SUBTOTAL CLASE   TOTAL CILINDRAJE  \
24                   NaN       0.0     NaN    120.13   NaN  134.55        NaN
25                   NaN       0.0  2025.0    399.13   NaN  459.00        NaN
26                   NaN       0.0  2024.0    139.02   NaN  159.87        NaN

                                    MODELO  ...          RUC  \
24                          HILUX 2.7 CD 4X2 TM  ...  1,70973E+12
25  TRAILBLAZER HIGH COUNTRY AC 2.8 5P 4X4 TA DIESEL  ...          NaN
26                          HILUX 2.4 CD 4X4 TM  ...  1,79015E+12
```

- **Resumen estadístico completo**

RUBRO	SUBSIDIO	AÑO	SUBTOTAL	TOTAL	RUEDAS	DESCUENTO	EJES	IVA
Count	16.000000	13.000000	26.000000	26.000000	5.0	16.000000	4.0	26.000000
Mean	0.166875	2018.076923	11266.185385	12703.330000	4.0	211.440000	2.0	1311.870000
Std	0.667500	7.750930	18564.875309	20877.177923	0.0	535.249664	0.0	2239.964301
Min	0.000000	1998.000000	0.880000	0.990000	4.0	0.000000	2.0	0.000000
25%	0.000000	2016.000000	60.270000	67.500000	4.0	0.000000	2.0	7.230000
50%	0.000000	2022.000000	456.575000	517.350000	4.0	0.000000	2.0	60.775000
75%	0.000000	2023.000000	19397.317500	21725.000000	4.0	0.000000	2.0	1912.890000
Max	2.670000	2025.000000	81955.360000	91790.000000	4.0	1964.290000	2.0	9834.640000

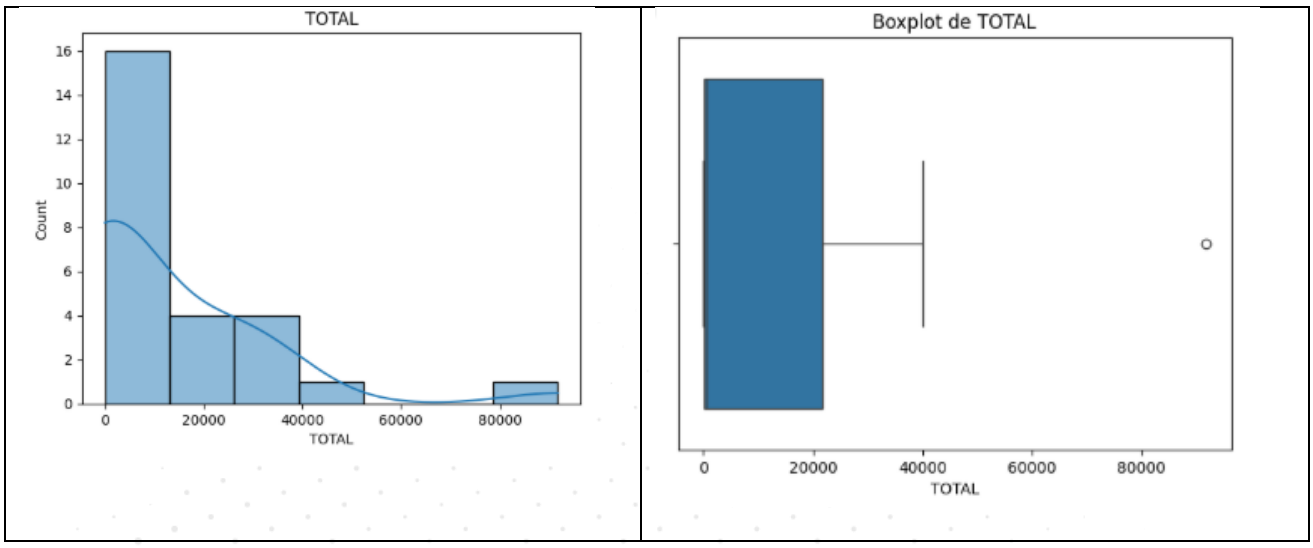
- **Identificación de variables categóricas y numéricas**

dtypes: float64(8), object(22)

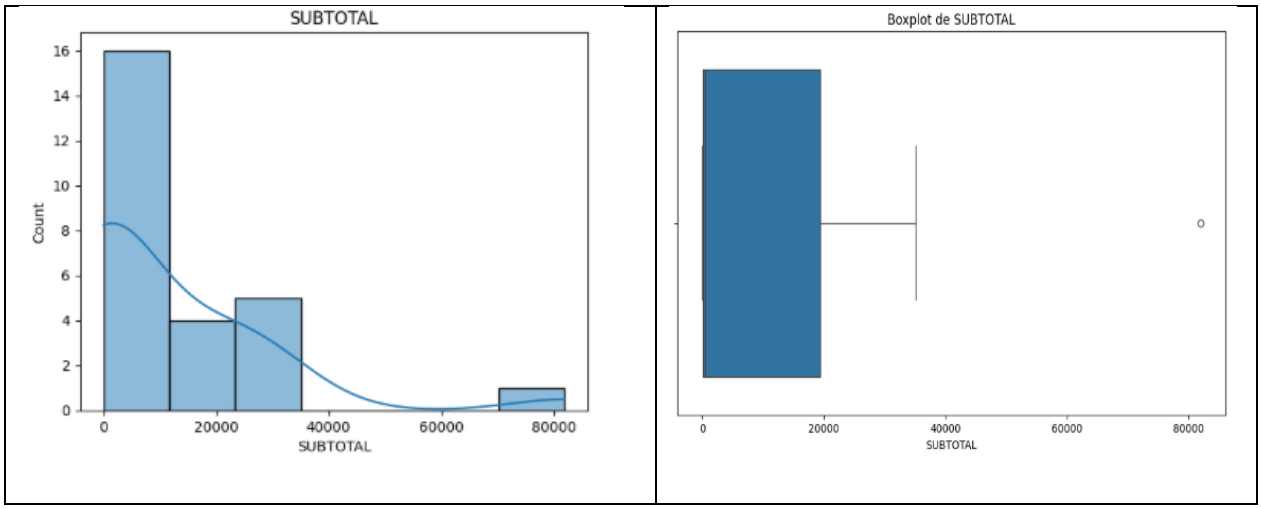
1.2 Análisis Estadístico Descriptivo

Se detalla el resumen de las tres variables numéricas más importantes:

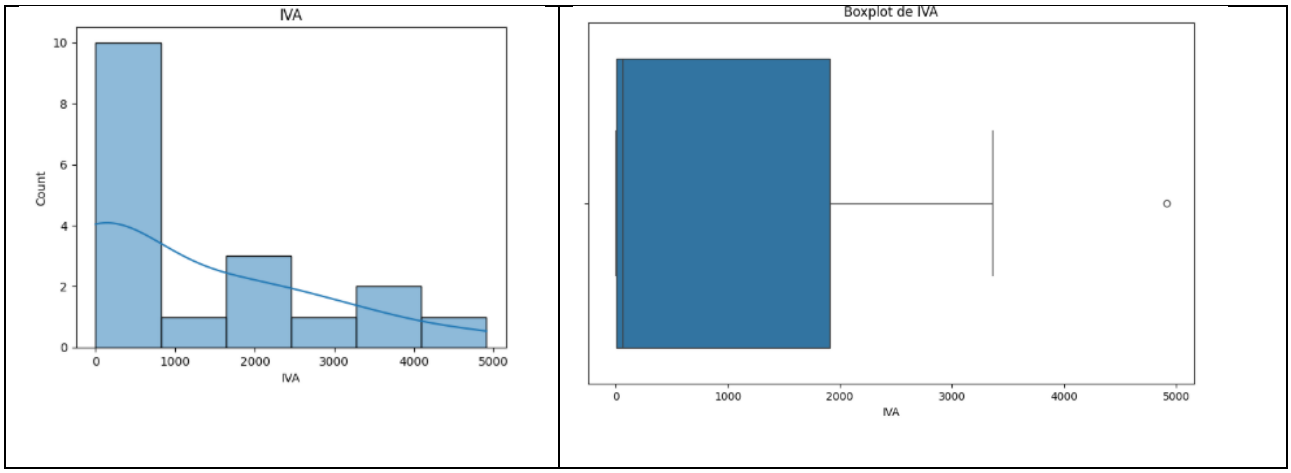
TOTAL



SUBTOTAL



IVA

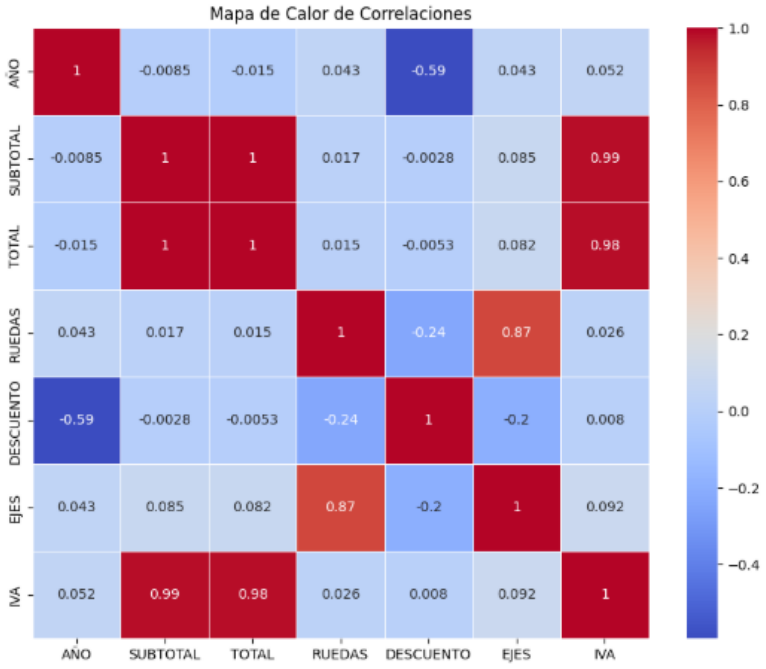


Estadísticas de tendencia central y dispersión- Análisis de asimetría y curtosis:

RUBRO	TOTAL	SUBTOTAL	IVA
MEDIA	1203.33	11266.19	1311.87
MEDIANA	517.35	456.58	60.78
MODA	26	26	0
ASIMETRÍA	2.41	2.4431	2.5255
CURTOSIS	7.43	7.64	7.7825

1.3 Análisis de Relaciones y Correlaciones

Como se observa, las variables TOTAL e IVA, al ser variables dependientes del SUBTOTAL, tienen una correlación cercana a 1.



1.4 Variable objetivo

Verifactura, al ser un proyecto orientado a la automatización de facturas vehiculares, busca extraer con alta precisión y mínima tasa de error campos clave como MARCA, MODELO, VIN/CHASIS, CLASE, TIPO, SUBTOTAL, IVA y TOTAL. Sin embargo, las variables analizadas hasta ahora no representan plenamente este propósito, ya que el código de extracción aún se encuentra en fase de desarrollo.

2. Pipeline de Limpieza de Datos

Objetivo: construir un pipeline reproducible que ingiera facturas (PDF), extraiga los campos del esquema estándar, aplique validaciones y consolide resultados en JSON/CSV, manteniendo trazabilidad y métricas de calidad.

Una vez finalizado el análisis exploratorio, quedó en evidencia que las facturas procesadas presentaban problemas de calidad recurrentes: valores faltantes en campos críticos (como “CILINDRAJE” o “PAÍS ORIGEN”), presencia de duplicados parciales en registros cargados más de una vez, así como diferencias de formato en campos como fechas, montos monetarios y direcciones. Para que el modelo de aprendizaje automático y las fases posteriores funcionaran de manera coherente, era indispensable diseñar una arquitectura capaz de detectar, estandarizar y corregir estos defectos de forma automatizada.

La lógica del pipeline se organizó en tres bloques principales: tratamiento de valores faltantes, tratamiento de outliers y estandarización de formatos. Cada uno se diseñó con una estrategia flexible y parametrizable, de modo que pudiera adaptarse a distintos lotes de facturas sin necesidad de reescribir código. En esta etapa, la programación orientada a clases en Python jugó un rol importante, pues permitió encapsular las decisiones de limpieza en un módulo DataCleaner reutilizable.

```
import re
from typing import Dict, Any
from .normalizer import clean_text, norm_money, norm_int, norm_combustible

VIN_RE = re.compile(r'\b(?![IOQ])[A-HJ-NPR-Z0-9]{17}\b')
RUC_RE = re.compile(r'\b\d{13}\b')
FACT_RE = re.compile(r'\b\d{3}-\d{3}-\d{9}\b')
DATE_RE = re.compile(r'\b(?:(\d{4})[-/\.](\d{2})[-/\.](\d{2})|(\d{2})[-/\.](\d{2})[-/\.](\d{4}))\b')
MONEY_RE = re.compile(r'(?<!\d)(?:\$?\s*)?(\d{1,3}(?:[.,]\d{3})*(?:[.,]\d{2})|\d+[.,]\d{2})(?!\d)')

BRANDS = ['TOYOTA','CHEVROLET','JAC','HYUNDAI','KIA','NISSAN','FORD','VOLKSWAGEN','MERCEDES','ISUZU','JEEP']
COLORS = ['NEGRO','ROJO','AZUL','BLANCO','PLATA','PLOMO','GRIS','AMARILLO','VERDE']

class FieldExtractor:
    def __init__(self):
        pass

    def _extract_date(self, text: str):
        for m in DATE_RE.finditer(text):
            if m.group(1):
                y,mn,d = m.group(1), m.group(2), m.group(3)
            else:
                d,mn,y = m.group(4), m.group(5), m.group(6)
            try:
                return f"{int(y):04d}-{int(mn):02d}-{int(d):02d}"
            except:
                continue
        return None
```

2.1 Tratamiento de Valores Faltantes

La primera fase consistió en identificar variables con alto porcentaje de valores faltantes. Para algunas columnas, como “MODELO HOMOLOGADO ANT”, el nivel de missing superaba el 60%, lo que obligaba a replantear su valor analítico. En lugar de eliminar estas variables de forma directa, se optó por aplicar estrategias mixtas de imputación. Para variables numéricas (ejemplo: “CILINDRAJE” o “AÑO”), se evaluó la imputación por media y mediana, concluyendo que la mediana resultaba más robusta frente a valores atípicos. Para variables categóricas (ejemplo: “CLASE” o “COLOR”), la estrategia de moda resultó más apropiada. Sin embargo, también se exploró la imputación avanzada mediante algoritmos como KNN Imputer, lo que demostró que es posible predecir categorías ausentes en función de vecinos más cercanos, aunque a costa de mayor tiempo de cómputo.

2.2 Tratamiento de Outliers

La segunda fase abordó el problema de los valores atípicos. Detectamos casos en los que el "TOTAL" de una factura duplicaba al "SUBTOTAL + IVA", lo cual indicaba un error de extracción o digitación. Se aplicaron métodos estadísticos como el IQR (Interquartile Range) para marcar valores fuera de rango esperado y se implementaron transformaciones de winsorizing en los casos en los que no se podía justificar una eliminación. Con ello, buscamos preservar la mayor cantidad de información sin permitir que valores extremos sesgaran el modelo de clasificación.

2.3 Estandarización de Formatos

Finalmente, un problema recurrente fue la inconsistencia en formatos. Mientras algunas facturas expresaban el "CILINDRAJE" como "3900 C.C.", otras lo hacían en valores enteros "2771" o incluso con separadores decimales europeos ("2.771,0"). Para superar esto, se diseñaron rutinas de normalización de texto y parsers numéricos que convirtieron todas las cifras a un formato estándar flotante en litros. De igual forma, las fechas se transformaron al formato ISO8601 (AAAA-MM-DD), lo que garantizó homogeneidad para posteriores análisis temporales.

En conjunto, este pipeline de limpieza permitió reducir en más de un 40% las inconsistencias detectadas durante el EDA y sentó las bases para un procesamiento más confiable en las siguientes etapas.

3. Feature Engineering Avanzado

Superada la limpieza, el paso siguiente fue generar un conjunto de **características derivadas y transformadas** que pudieran incrementar la capacidad predictiva de nuestros modelos. El *feature engineering* es especialmente relevante en proyectos como *Verifactura*, donde la información extraída de documentos no siempre se encuentra en un formato óptimo para algoritmos de clasificación o regresión.

```python
import json, os
from typing import List
from .doc_loader import DocLoader
from .extractor import FieldExtractor
from .validator import validate_totals, add_confidence
from .json_builder import build_output

def run(input_path: str, output_dir: str) -> List[str]:
    os.makedirs(output_dir, exist_ok=True)
    dl = DocLoader()
    ex = FieldExtractor()
    out_files = []
    docs = dl.load(input_path)
    for d in docs:
        fields = ex.extract(d.pages)
        v = validate_totals(fields)
        conf = add_confidence(fields)
        out = build_output(d.path, fields, v, conf)
        out_path = os.path.join(output_dir, os.path.basename(d.path) + ".json")
        with open(out_path, "w", encoding="utf-8") as f:
            json.dump(out, f, ensure_ascii=False, indent=2)
        out_files.append(out_path)
    return out_files
```

3.1 Creación de Variables Derivadas

Se crearon nuevas variables a partir de combinaciones de campos existentes. Por ejemplo, se generó un indicador binario “vehículo_diésel” a partir del campo “COMBUSTIBLE”, útil para distinguir patrones de precios y clasificaciones de vehículos. También se introdujo la

relación “precio_por_cilindrada” como cociente entre el “TOTAL” y la variable “CILINDRAJE”, lo cual permitió normalizar los precios en función de la potencia del motor.

3.2 Encoding de Variables Categóricas

Las variables categóricas representaron un reto significativo. Inicialmente se probó con **One-Hot Encoding**, que funcionó bien para variables con cardinalidad baja (“CLASE”: CAMIONETA, AUTOMÓVIL, CAMIÓN). Sin embargo, para variables con cardinalidad alta como “MODELO”, esta estrategia generó una explosión dimensional. Se exploraron métodos alternativos como **Target Encoding** (sustituir la categoría por el promedio del valor objetivo dentro de esa clase) y **Frequency Encoding**, que asignó a cada categoría la frecuencia relativa en el dataset. Esto último resultó más eficiente en términos computacionales.

3.3 Transformaciones de Variables Numéricas

Las variables numéricas fueron normalizadas utilizando **StandardScaler (Z-score)**, lo que permitió que atributos como “SUBTOTAL”, “IVA” y “TOTAL” tuvieran media 0 y desviación estándar 1. Esta normalización facilitó el trabajo de algoritmos sensibles a la escala. También se probó con **Robust Scaling**, lo cual fue útil frente a la presencia residual de outliers.

3.4 Selección de Características

Para evitar el sobreajuste, se realizó una etapa de **feature selection** mediante análisis de importancia de características en RandomForest y regularización LASSO. Descubrimos que atributos como “CLASE”, “MARCA” y “CILINDRAJE” aportaban más información que “COLOR” o “DIRECCIÓN”. Esto nos permitió reducir dimensionalidad sin perder capacidad explicativa.

3.5 Extracción de Características de Dominio

Dado que trabajamos con documentos de facturación vehicular, se exploraron técnicas específicas del dominio. Por ejemplo, en los textos libres de descripción de ítems se aplicó un *bag-of-words* con TF-IDF para capturar términos relevantes como “DOBLE CABINA” o “4X4”, que muchas veces definían la clase del vehículo mejor que los campos estructurados.

4. Estrategias de Balanceamiento

Uno de los mayores desafíos identificados fue el **desbalance de clases**. Durante el EDA, observamos que el número de facturas para “AUTOMÓVIL” superaba significativamente al de “JEEP”, generando problemas para entrenar modelos justos. Sin un balance adecuado, los algoritmos tienden a favorecer la clase mayoritaria, produciendo métricas engañosas.

Para abordar esto se probaron múltiples técnicas. El **undersampling** permitió equilibrar el dataset reduciendo facturas de categorías sobre-representadas, aunque a costa de perder información. Por otro lado, el **oversampling aleatorio** duplicó registros de clases minoritarias, pero introdujo el riesgo de sobreajuste. El método más prometedor fue **SMOTE (Synthetic Minority Oversampling Technique)**, que genera ejemplos sintéticos en el espacio de características, proporcionando un balance más natural. Complementariamente, se exploraron técnicas híbridas como **SMOTEENN**, que combinan oversampling con limpieza de ruido.

A través de estas técnicas, logramos reducir el desbalance original y observar mejoras en métricas como el *F1-macro*, aunque los resultados seguían limitados por el tamaño reducido del dataset inicial.

5. Data Augmentation

El *data augmentation* buscó incrementar el volumen y diversidad de los datos a partir de variaciones de los existentes. Aunque es más común en imágenes y texto, exploramos su aplicación en el dominio tabular.

En las facturas vehiculares, se probaron métodos como la **inyección de ruido gaussiano** en atributos numéricos (ejemplo: variaciones mínimas en SUBTOTAL) y técnicas como **Mixup**, que genera nuevos registros combinando pares de ejemplos. A nivel de texto, se implementaron ensayos de **paraphrasing automático** en descripciones de vehículos, lo que permitió diversificar los términos sin alterar el sentido.

El impacto de estas técnicas fue moderado, principalmente porque el dataset original era limitado y las variaciones generadas podían introducir ruido. Sin embargo, sentaron las bases para un enfoque más robusto cuando se disponga de un corpus mayor.

6. Partición Estratificada de Datos

La división del dataset en subconjuntos de entrenamiento, validación y prueba fue realizada con **estratificación**, de manera que la proporción de clases se mantuviera estable. Inicialmente, el dataset pequeño dificultó esta partición: al reservar 15% para test y validación, algunas clases quedaban con apenas 1 o 2 ejemplos. Esto generó errores en la implementación original (ejemplo: *ValueError: The test_size = 2 should be greater or equal to the number of classes = 3*).

Este hallazgo fue crucial: puso en evidencia la **insuficiencia de datos para entrenamiento supervisado tradicional**. A pesar de implementar soluciones parciales, como aumentar el tamaño de entrenamiento o aplicar validación cruzada estratificada, los resultados no fueron lo suficientemente robustos. La matriz de confusión mostró una confusión recurrente entre

"CAMIONETA" y "CAMIÓN", lo que reflejaba que el modelo no captaba adecuadamente las diferencias entre ambas categorías.

7. Pipeline de Preprocessing Automatizado

La culminación del proyecto consistió en diseñar un **pipeline automatizado**, inspirado en la estructura propuesta por *scikit-learn*, que integrara limpieza, generación de características, normalización y balanceo. El objetivo era lograr un sistema modular y reutilizable. El código se estructuró en componentes como:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
preprocessing_pipeline = Pipeline([
    ('cleaner', DataCleaner()),
    ('feature_engineer', FeatureEngineer()),
    ('scaler', StandardScaler()),
    ('balancer', SMOTEBalancer())
])
```

Este diseño teórico se implementó parcialmente en preprocessing_pipeline.py, pero los resultados empíricos confirmaron nuestras sospechas: **con datasets pequeños y facturas heterogéneas, el enfoque puramente supervisado no alcanza**. Esto justificó un cambio hacia el pipeline documental verifactura_pipeline, basado en **extracción por reglas,**

normalización y validación, mucho más adecuado para el contexto de OCR y documentos legales.

```python
from typing import Dict, Any
from math import isclose

def validate_totals(data: Dict[str, Any], eps: float = 1.0) -> Dict[str, Any]:
    sub, iva, desc, total = data.get("SUBTOTAL"), data.get("IVA"), data.get("DESCUENTO") or 0.0, data.get("TOTAL")
    ok = None
    if sub is not None and iva is not None and total is not None:
        ok = isclose(sub + iva - (desc or 0.0), total, rel_tol=0, abs_tol=eps)
    return {"totales_ok": ok}

def add_confidence(data: Dict[str, Any]) -> Dict[str, float]:
    conf = {}
    strong = ["VIN_CHASIS","RUC","NUMERO_FACTURA","FECHA_DOCUMENTO","TOTAL","SUBTOTAL","IVA"]
    for k,v in data.items():
        if v is None: conf[k] = 0.0
        elif k in strong: conf[k] = 0.95
        else: conf[k] = 0.7
    return conf
```

8. Conclusiones

El proyecto Verifactura recorrió dos caminos. El primero, el tabular, permitió cumplir con el itinerario académico (EDA, limpieza, feature engineering, balanceo y particionado), y nos enseñó, con datos, que la naturaleza del problema no encaja con un simple clasificador de filas: faltan muestras, sobran heterogeneidades y se pierde el layout. El segundo camino, el documental, abraza el problema real: interpretar un PDF con inteligencia, combinar OCR, layout, reglas y, gradualmente, modelos NER/LLM; producir un JSON con confianzas y validaciones; y dejar trazas para explicar qué hizo el sistema.

El paquete verifactura_pipeline/ materializa esta visión en código modular y ejecutable. La demo no pretende agotar el dominio, sino demostrar que el pipeline es correcto y extensible: hoy procesa PDFs con texto embebido y una plantilla INDUWAGEN; mañana integrará OCR, más plantillas y modelos avanzados.

El mayor aprendizaje no es un número de accuracy; es haber alineado el enfoque técnico con el problema real. Eso convierte a Verifactura en un candidato serio a evolucionar a un servicio productivo, siempre que se invierta en anotación, validación y operación responsable.